

FREE Smart Modbus Master

Contents

DEFINITIONS.....	1	MODBUS MASTER ASSISTED CONFIGURATION.....	4
GENERAL DESCRIPTION	2	MODBUS CUSTOM EDITOR.....	8
REQUIREMENTS	2	'MANUAL' PROGRAMMING	11
FREE SMART MODBUS MASTER MODELS	2	APPENDIX - EXAMPLES OF MANUAL PROGRAMMING.....	13
RS485 SERIAL.....	3	APPENDIX - LIBRARIES.....	17
FIRST START-UP	3		
ENABLE MODBUS MASTER.....	4		

DEFINITIONS

- **BIOS** is synonymous with **Firmware** in **FREE Studio**
- **FREE Smart Modbus:** in this document, understood to mean **FREE Smart Modbus Master**
- **Modbus:** understood to mean the **Modbus® RTU** version
- **Target** is synonymous with **Smart** in **FREE Studio**
- **Target blocks:** the **default functions library** includes functions specific to the target device **FREE Smart Modbus**

Important note for developers using FREE Studio:
the '**Library**' includes a target blocks chart containing the functions to be used for managing **Modbus** functionalities.
(See **Appendix - Libraries** and **FREE Studio** manual for details).

For further information on **FREE Studio** functionalities not expressly tied to the **Modbus Master** configuration, consult **Help On Line** (press key **F1**) from the **FREE Studio** work environment, or the **FREE Studio Quick Start** aid, which is downloadable from the Eliwell site (registration required).



GENERAL DESCRIPTION

REQUIREMENTS

The **FREE Smart Modbus Master target¹** is available in the **FREE Studio** development environment from version 3.1.0 onwards.

The firmware mask is number **542.***

The programming development environment is **FREE Studio Application**

FREE SMART MODBUS MASTER MODELS

The **FREE Smart Modbus Master target** is available in 4Din versions with display:

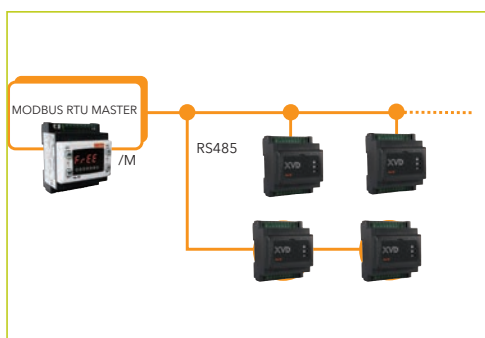
	Model	p/n	Notes
	FREE Smart 12...24V~ (/M)	SMD5500/C/S/M	4Din with display
	FREE Smart 4500 100..240V~ (/M)	SMD4500/C/S/M	4Din with display

Other p/ns available on request. Contact Eliwell Sales Office

Modbus Master models are identified by the suffix **/M** and are provided with on-board RS485 serial interface (suffix **/S**)

Modbus Master models can be connected to the slave nodes of Eliwell tools or those of third parties.

Modbus messages are identified by the relative functions (**target blocks**) implementing the Modbus commands that will be sent over the bus, as described in Appendix- Libraries



¹ in FREE Studio, the target is indicated as FreeSmart Modbus

RS485 SERIAL



There is only one RS485 serial communication interface: connection to the target is not possible with Modbus RTU active.

If the **target** is empty, the serial connection will be set to **Slave** mode.

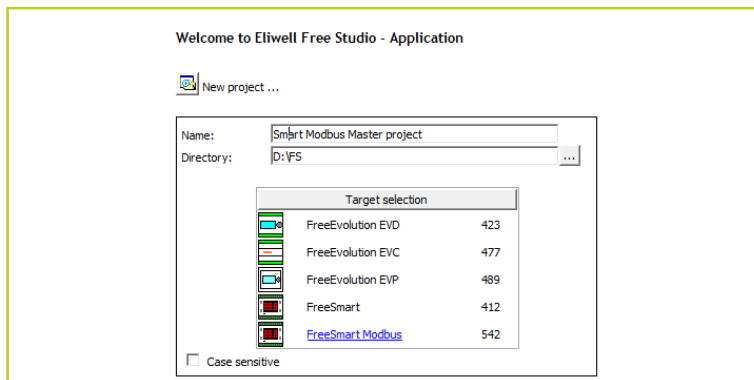
If there is a PLC application running, there will be a pause of 5 seconds before execution occurs, during which a connection can be made through **FREE Studio** by forcing Slave mode. Once the 5 seconds have elapsed without any connection to the target, the serial connection defaults to **Modbus Master** mode

In any event, Modbus Master mode can always be enabled/disabled by means of the IEC code, using the relative function `sysMbMRTU_Enable()` described in [Appendix - Libraries](#)

FIRST START-UP

1. Click the **Launch FREE Studio** icon located on the Desktop
2. Click the **FREE Studio Application** icon.

The program appears as illustrated below:



The following options will now be available:

- Create a new project '**New Project ...**'
 - the program will prompt selection of the FreeSmart or the FreeSmart Modbus target
 - in this case select FreeSmart **Modbus**
 - the numbers at the side indicate the firmware version (mask)
- Open an existing project '**Open project ...**'
- Open recent projects '**Open an existing project ...**' by selecting from a list.

Existing projects have the extension **.ppjs**



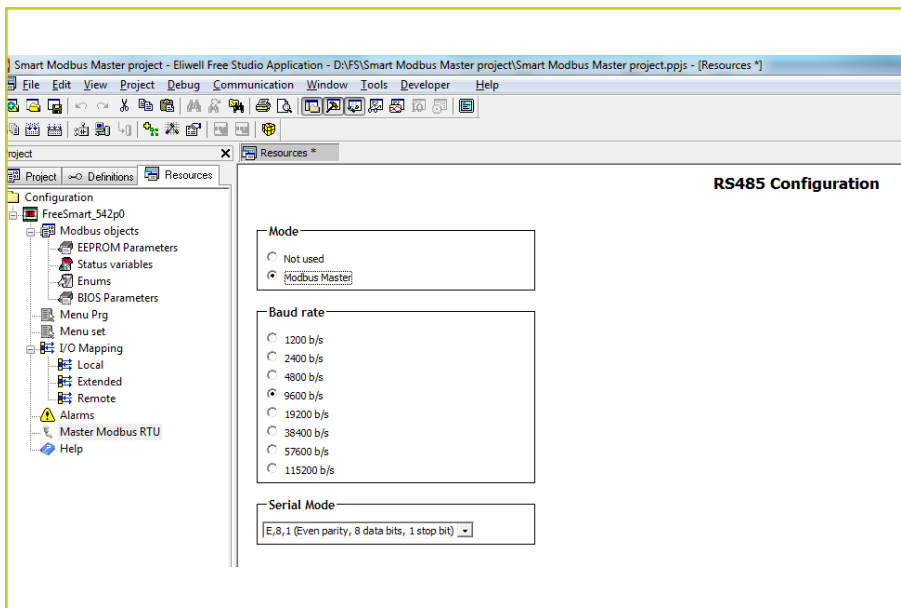
ENABLE MODBUS MASTER

FREE Studio offers two ways of enabling Modbus Master

1. using assisted configuration².
2. using the library function `sysMbMRTU_Enable()`³

MODBUS MASTER ASSISTED CONFIGURATION

Click the Resources tab, and select **Modbus Master**



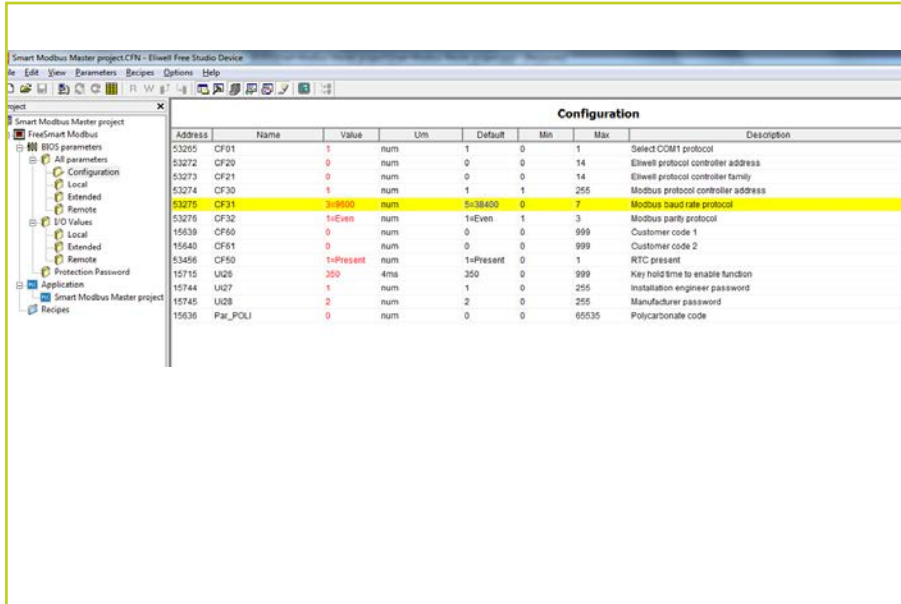
When **Modbus Master** is selected, it will be possible to set Baud Rate⁴ and parity (**Serial Mode**). Bear in mind that the configuration procedure impacts on the configuration of BIOS parameters **CF31** e **CF32**.

CF31 and **CF32** are visible

- from the menu **Resources > Modbus Objects > BIOS Parameters**, if they are added to the list with Add, or
- from **FREE Studio Device**: from the drop-down menu, select **Developer > Open with FREE Studio Device** and access the **Configuration** folder
- from the **BIOS** menu of the **target** (press **esc+set (Prg)** and access the folder **PAr > CF**)

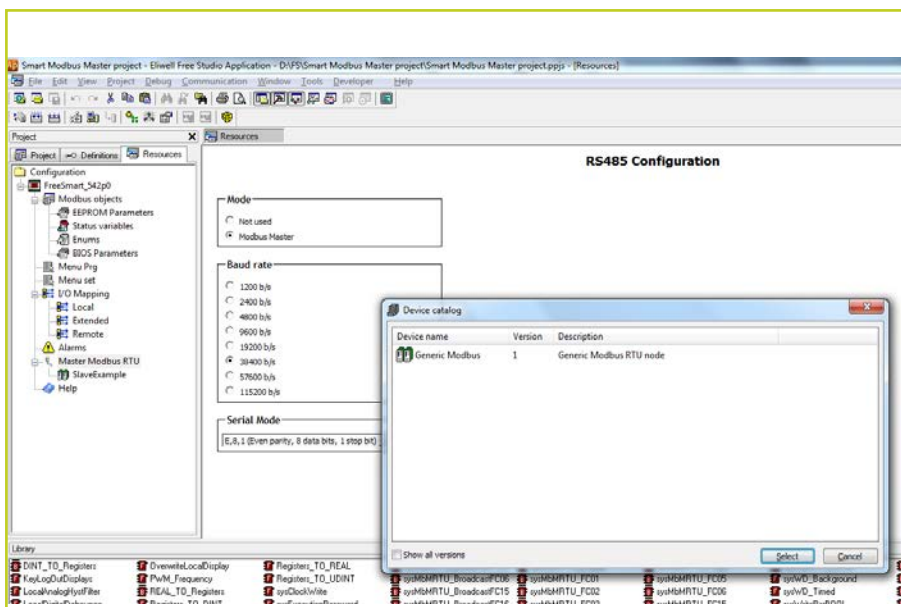
² defined by Help On Line as **CONFIGURATION WIZARD**
³ see **Appendix - Libraries**
⁴ operation is not guaranteed at rates above 19200 b/s

BIOS CF31 and CF32 parameters. Note the modified default (CF31: 3=9600 → 5=38400)

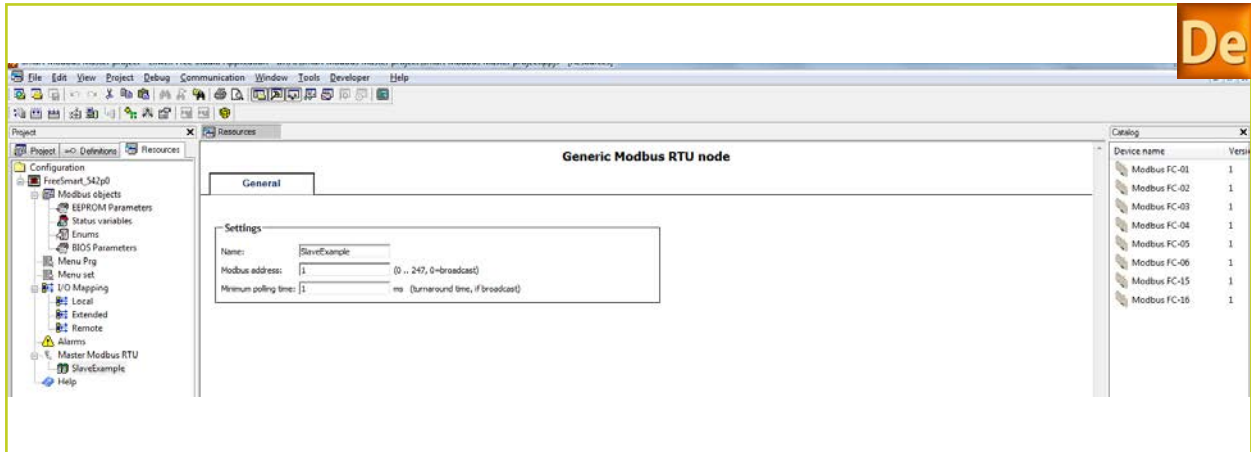
Address	Name	Value	Um	Default	Min	Max	Description
53205	CF01	1	num	1	0	1	Select COM1 protocol
53272	CF20	0	num	0	0	14	Eliwell protocol controller address
53273	CF21	0	num	0	0	14	Eliwell protocol controller family
53274	CF30	1	num	1	1	255	Modbus protocol controller address
53275	CF31	3=9600	num	5=38400	0	7	Modbus baud rate protocol
53276	CF32	1=Even	num	1=Even	1	3	Modbus parity protocol
15638	CF60	0	num	0	0	999	Customer code 1
15640	CF61	0	num	0	0	999	Customer code 2
53456	CF50	1=Present	num	1=Present	0	1	RTC present
15715	UI26	350	4ms	350	0	999	Key hold time to enable function
15744	UI27	1	num	1	0	255	Installation engineer password
15745	UI28	2	num	2	0	255	Manufacturer password
15636	Par_POLI	0	num	0	0	65535	Polycarbonate code

Right clicking on **Resources > Modbus Master RTU** and selecting **Add** from the pop-up menu, a generic slave node **Generic Modbus** can be added



Double clicking the node, the **Settings** pane will appear to allow selection of the node settings





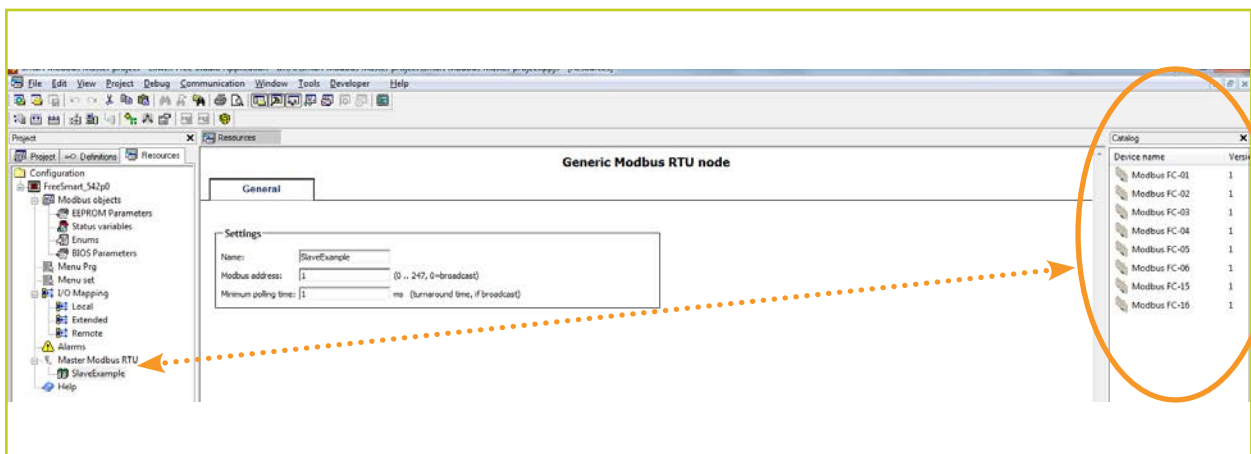
Besides the name⁵, the settings available are:

Modbus Address: Modbus slave address. If address = 0, the node is virtual (no device connected) and is utilized for broadcast messages

Minimum polling time: minimum polling time for sending messages addressed to the single node. If the address is 0, this is the **broadcast messages turnaround** time (the interval between one broadcast message and the next).

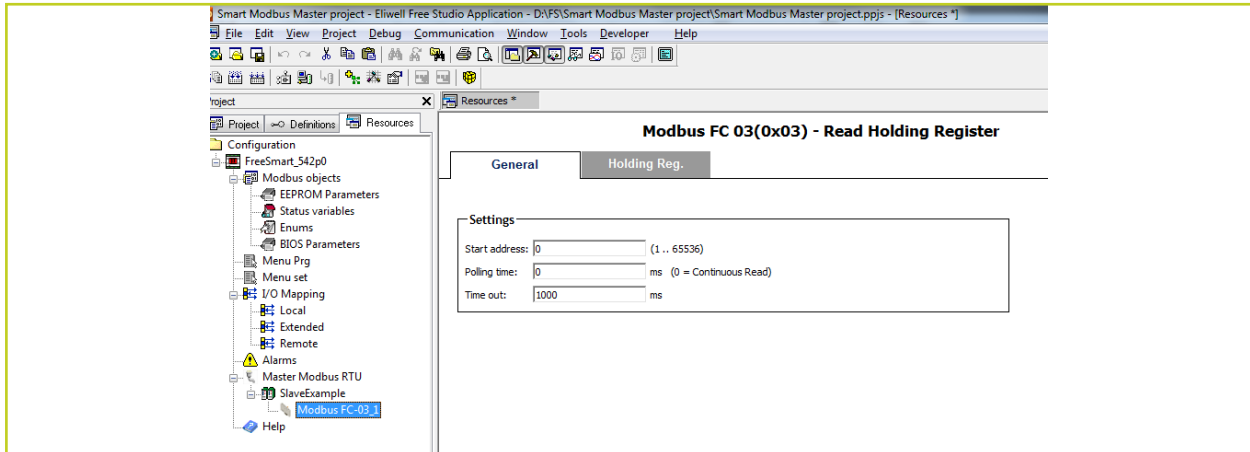
Note. With assisted configuration, it is not sufficient simply to add a node; one or more Modbus messages⁶ must be added.

Select and drag the desired message from the **Catalog** pane, as illustrated:



⁵ in the example, the default instance is Generic Modbus_1, Generic Modbus_2 etc.
⁶ described in Appendix

Double clicking the node, the **Settings** pane will appear to allow selection of the node settings. The pane has 2 tabs, **General** and **Object**⁷ (in the example illustrated, Object = Holding Reg.)



General Pane Settings

Besides the name, the settings available are:

Start Address: address of the first Modbus slave object.

The range is 1...65536. Bear in mind that some manufacturers use 0...65535.

In this case **+1** must be added to the Start Address

Polling time: the time within which the message will be processed, expressed in ms.

If time = 0:

write: only if the value is changed

read: indicates the maximum speed

“Object” Pane Settings

This second tab varies according to the message (note 6) and defines the list of readable or writeable objects.

Similarly, in the case of Modbus objects (EEPROM parameters, BIOS Status Variables, etc.)⁸ there is the option of adding or removing up to

	Modbus objects
8 registers	16-bit
16 Coil/Discrete Inputs	1-bit

⁷ depending on the Modbus message, the Object tab = Holding Reg, Inputs, Coil, Single Reg. etc
⁸ see Help On Line (key F1) or FREE Studio Quick Start

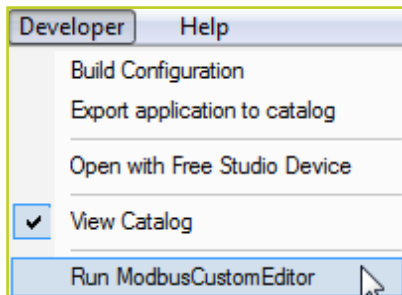


MODBUS CUSTOM EDITOR

With the Editor, **custom** Modbus devices can be created directly by the user.

When these devices are used in the project and added to the catalogue, they can then be reused conveniently in third party Modbus slave devices. Implementation of the Modbus map is carried out once only, thereby simplifying future utilizations as there is no need to create new Modbus messages and functions.

To create a custom Modbus device, select **Developer > Run ModbusCustomEditor** from the drop-down menu



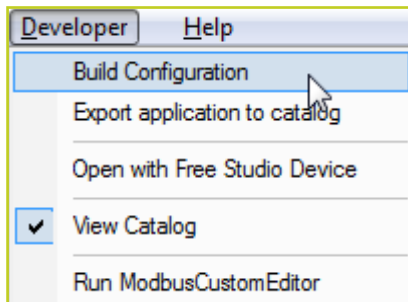
The external ModbusCustomEditor work environment will launch with the creation of a blank document: **'empty'**⁹

⁹ details on creating a custom Modbus device can be found in the document entitled **FREE Studio Connection User Manual**, available on the Help pages of the FREE Studio Connection work environment (key F1)



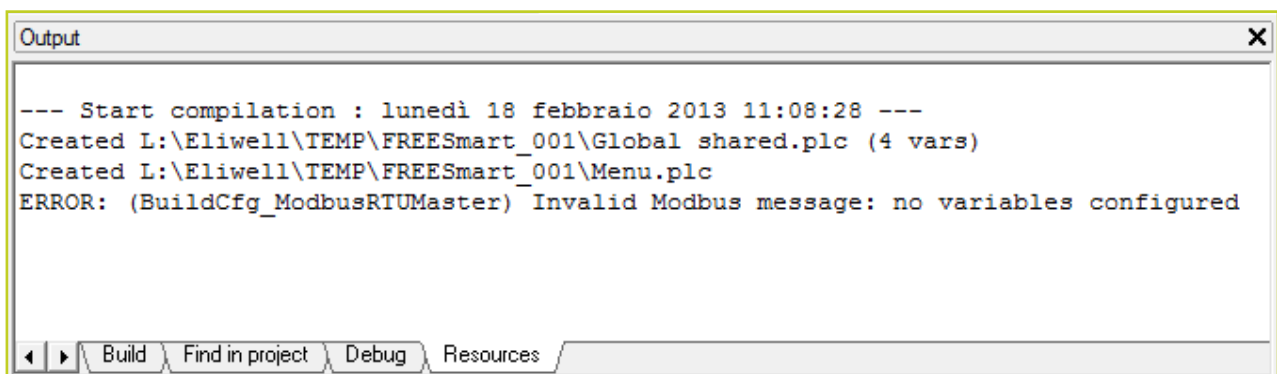
Building Configuration

The **Modbus Master** configuration is built automatically on saving the project. It will be possible to 'force' the FREE Studio Application to create the configuration, by selecting **Developer > Build configuration** from the drop-down menu



Configuration errors

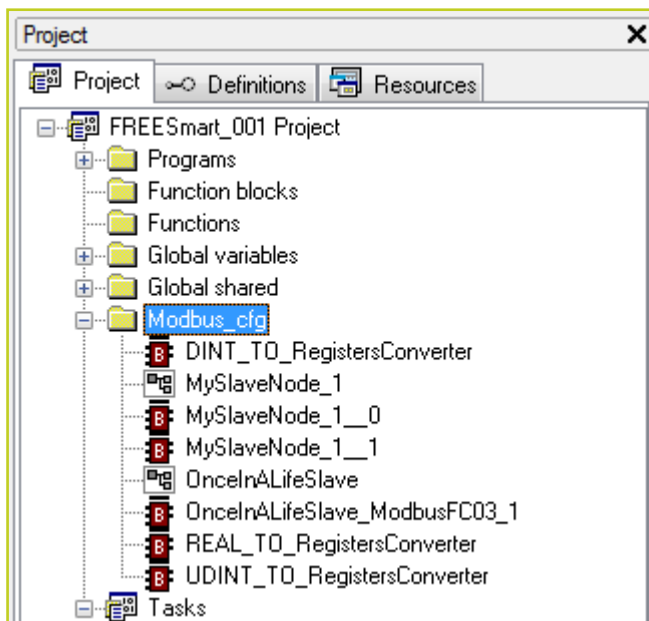
In the event that configuration errors are detected, an error message appears in **Output > Resources**



Double-clicking on the error message, FREE Studio Application re-addresses to the relative page of the configuration wizard, which will resolve the issue without delay.

Modbus CFG

Once the configuration has been concluded without error, the **Project** pane will display a folder named **Modbus_cfg**, containing the elements generated by the Modbus Master configuration wizard.



Also, there is always an IEC 61131-3 STRUCT instance for each single node configured as a slave, and a function block instance for each single Modbus message.

These elements are exploited by the PLC application and by debug operations, which access the resources as illustrated in the next chapter on 'manual' programming



'MANUAL' PROGRAMMING

As an alternative to assisted configuration, the developer can create the **IEC 61131-3** code using **Modbus Master** functions.

Assisted configuration allows the creation of nodes and messages (e.g. readable and/or writeable Expansions and I/O values); polling of these slaves takes place cyclically as a background task.

Should the intention be to interrogate a resource just once, or on the basis of a specific event determined by way of a PLC code, it will be preferable to write dedicated code to meet the explicit requirements of the developer.

Association of the Modbus program with a Task

For the **Modbus** program to run, it must be associated with a '**task**'.

There are various types of task:

- **Boot.** Task executed once only on starting up the system
- **Init.** Task executed at each download of the application and on starting up the system (after **Boot**)
Note. The associated program initializes slaves and messages according to the configuration, with fixed values that are independent of the run time
- **Timed.** Task executed at regular intervals, which can be set by the developer. Default setting every 100ms.
Note. Modbus messages do not interfere with this task
- **Background.** Task executed as low priority (once or repeatedly, between the end of one **Timed** task and the start of the next).

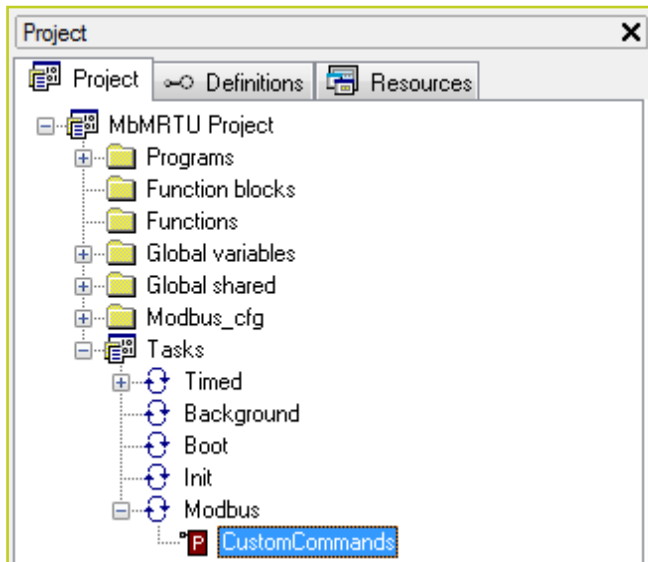
Also available, in addition to these, is the **task¹⁰**:

- **Modbus.** Task executed to implement **Modbus Master**, calling relative function blocks, and to send messages.

Note. Each new project has the main program associated with the background task (the main program can still be eliminated and/or associated with other tasks).

¹⁰ firmware mask 542 only.*

To activate a task, point the mouse over the name, right-click and select **Add program**



The PLC Modbus task is an invaluable resource for the developer: it encodes part of the Modbus RTU master configuration manually: this is a dedicated 'entry point' needed for every operation that produces a message to be sent over the bus.

Using this rule, Modbus messages will not influence the timed PLC task and will have only a limited impact on low priority tasks (asynchronous mode).

Note. The dedicated code combines with the code generated by the **configuration wizard** to determine an increment of the code and the time values of the Modbus Master (interrogation on bus, etc)



APPENDIX - EXAMPLES OF MANUAL PROGRAMMING

Example 1 Send single message 'just once'

Sending a message to a slave node just once, immediately after power-up.

The message is a Modbus 05 function; the target is already managed by the guided configuration procedure (wizard) (this implies that there is already an existing STRUCT instance available in the project for the slave node).

A program can be added to the PLC Modbus task with the following instructions:

```
(* If Modbus master is not enable, skip custom messages *)
IF NOT sysMbMRTUStatus.enabled THEN

    (* When Modbus master is enabled again, send the message again *)
    sent := FALSE;
    RETURN;
END_IF;

(* Una tantum message to Slave #2 *)
IF NOT sent THEN

    (* Send the message *)
    Slave2_WriteCoil12.node := ADR( Slave2 );
    Slave2_WriteCoil12.address := 12;
    Slave2_WriteCoil12.timeout := 1000;
    Slave2_WriteCoil12( object := TRUE );

    (*
    * Check if correctly received by the slave:
    * in this case never send it again.
    *)
    sent := NOT Slave2_WriteCoil12.error;
END_IF;
```

Note.

The boolean type Sent variable is defined as a local variable

The Slave2_WriteCoil12 variable is created as an instance of the target block

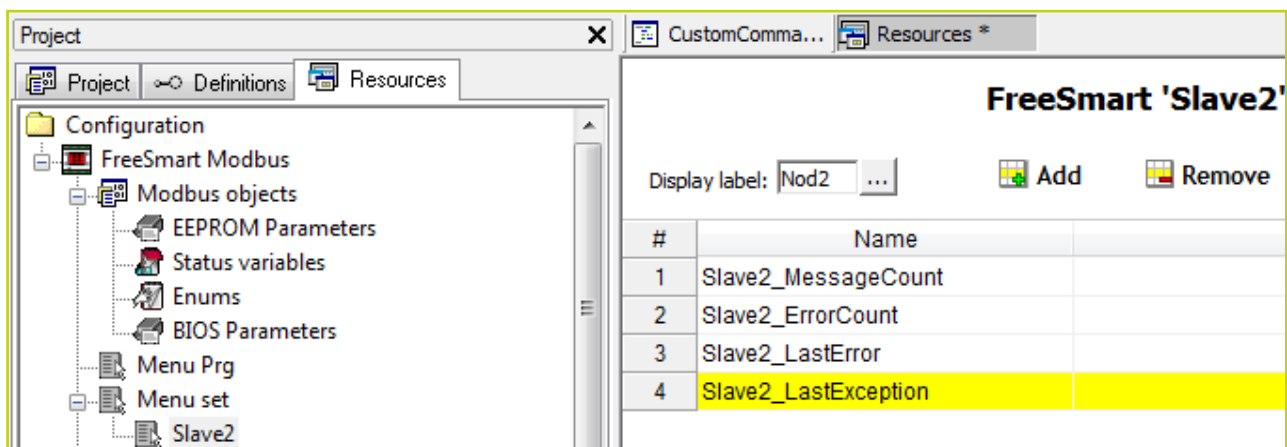
sysMbMRTU_FC05.

Example 2 Creation of diagnostics menu

The code can also be used for the publication of diagnostics information relating to the Modbus Master in a FREE Smart menu.

The intention is to publish the diagnostics counters for a specific slave node: these must appear in a submenu that the user can access via a local display from the Set menu.

The first step is to set all of the status variables required, and assemble them in a menu:



```
(* Publish Slave #2 diagnostics counters *)
Slave2_MessageCount := Slave2.diagnostics.messageCount;
Slave2_ErrorCount := Slave2.diagnostics.errorCount;
Slave2_LastError := Slave2.diagnostics.lastError;
Slave2_LastException := Slave2.diagnostics.lastException;
```

Example 3 Disable Modbus Master by keystroke


The user may wish to disable the Modbus RTU Master by pressing and holding the ESC key of the target (local display), to allow the debug of FREE Studio applications for collection to the target device, even if the PLC application has already been launched. In symmetrical manner, the Modbus Master is reactivated by pressing and holding the SET key.

```
(* On ESC key long-press, disable Modbus RTU master *)
IF sysKeyFunctions[ 1 ] THEN

    (* Reset long-press event *)
    sysKeyFunctions[ 1 ] := FALSE;

    (* Disable Modbus RTU master *)
    boolWarningsKiller := sysMbMRTU_Enable( FALSE );

END_IF;

(* On SET key long-press, enable Modbus RTU master *)
IF sysKeyFunctions[ 3 ] THEN

    (* Reset long-press event *)
    sysKeyFunctions[ 3 ] := FALSE;

    (* Enable Modbus RTU master *)
    boolWarningsKiller := sysMbMRTU_Enable( TRUE );

END_IF;
```

Note `sysKeyFunctions` is a function available in the **Library > Target variable** section of **FREE Studio Application** as described in **FREE Studio**

Target Variables

`sysKeyFunctions`

Type : VARIABLE
 Attribute : Normal
 Val type : BOOL
 Address : %MB23.0

Long press of Key functions, when a key is released the related flag is not automatically reset.
 [0]=F1 [1]=F2 [2]=F3 [3]=F4

PS [0]=UP [1]=Esc [2]=Down [3]=Set



Example 4 Disable slave node

The user wishes to verify whether or not a slave node is responding (or no longer responding), and if not, to stop further messages being sent to the node. In effect, the time that elapses (typically a few seconds) when waiting for a response from an unresponsive slave (until the end of the timeout period), influences the overall performance of the Master.

The following code checks the number of errors that have occurred and disables the slave node if the number exceeds a given threshold.

```
(* Test if the user issued a command to enable Slave #2 again *)
slaveEnableRisingEdge( Slave2.enabled );
IF slaveEnableRisingEdge.q THEN
    (*
    * Reset the counters, so that if the slave is enabled
    * again (with an explicit command available as a menu
    * item) the test below works again.
    *)
    Slave2.messageCount := 0;
    Slave2.errorCount := 0;
    Slave2.lastError := ErrNone;
    Slave2.lastException := ExcNone;
END_IF;

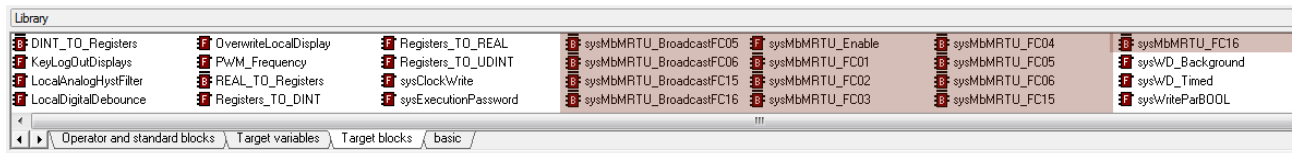
(* Is the error count above the configured threshold? *)
IF Slave2.diagnostics.errorCount > MAX_ERRS THEN

    (* Disable Slave #2 *) Slave2.enabled := FALSE;

END_IF;
```

APPENDIX - LIBRARIES

Indicated below are the Modbus Master functions available in the **Library > Target blocks** section of **FREE Studio Application** as described in **FREE Studio**



Enable

sysMbMRTU_Enable

Enable/Disable Modbus master: that allows to dynamically switch to programming/debug mode

Type : FUNCTION

Code type : EMBEDDED

Input vars num: 1

enable : BOOL (* If TRUE, enables Modbus master; otherwise, disables it *)

Result type : BOOL

Read/write

sysMbMRTU_FC01

Read Coils - Function 01 (0x01)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 5

node : @MbMRTUNode (* Slave node *)

base : UINT (* Address of the first Coil to be read *)

no : UINT (* Number of contiguous Coils to be read *)

pollingTime : UINT (* Polling time [ms] *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 18

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

object1 : BOOL (* 1st Coil *)

...

object16 : BOOL (* 16th Coil *)

Extern vars num: 0



sysMbMRTU_FC02

Read Discrete Inputs - Function 02 (0x02)



Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 5

node : @MbMRTUNode (* Slave node *)

base : UINT (* Address of the first Discrete Input to be read *)

no : UINT (* Number of contiguous Discrete Inputs to be read *)

pollingTime : UINT (* Polling time [ms] *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 18

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

object1 : BOOL (* 1st Discrete Input *)

...

object16 : BOOL (* 16th Discrete Input *)

Extern vars num: 0

sysMbMRTU_FC03

Read Holding Registers - Function 03 (0x03)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 5

node : @MbMRTUNode (* Slave node *)

base : UINT (* Address of the first Holding Register to be read *)

no : UINT (* Number of contiguous Holding Registers to be read *)

pollingTime : UINT (* Polling time [ms] *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 10

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

object1 : INT (* 1st Holding Register *)

...

object8 : INT (* 8th Holding Register *)

Extern vars num: 0



sysMbMRTU_FC04

Read Input Registers - Function 04 (0x04)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num : 5

node : @MbMRTUNode (* Slave node *)

base : UINT (* Address of the first Input Register to be read *)

no : UINT (* Number of contiguous Input Registers to be read *)

pollingTime : UINT (* Polling time [ms] *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 10

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

object1 : INT (* 1st Input Register *)

...

object8 : INT (* 8th Input Register *)

Extern vars num: 0

sysMbMRTU_FC05

Write Single Coil - Function 05 (0x05)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num : 5

node : @MbMRTUNode (* Slave node *)

address : UINT (* Address of the Coil to write *)

object : BOOL (* Coil value *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0

i n v e n s y s
Controls

sysMbMRTU_FC06

Write Single Register - Function 06 (0x06)



Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 5

node : @MbMRTUNode (* Slave node *)

address : UINT (* Address of the Holding Register to write *)

object : INT (* Holding Register value *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0

sysMbMRTU_FC15

Write Multiple Coils - Function 15 (0x0F)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num : 21

node : @MbMRTUNode (* Slave node *)

base : UINT (* Address of the first Coil to write *)

no : UINT (* Number of contiguous Coils to write *)

object1 : BOOL (* 1st Coil value *)

...

object16 : BOOL (* 16th Coil value *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0



FREE Application Notes

code 9IS24300-1 - rel. 13.03.13

sysMbMRTU_FC16

Write Multiple Registers - Function 16 (0x10)



Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 13

node : @MbMRTUNode (* Slave node *)

base : UINT (* Address of the first Holding Register to write *)

no : UINT (* Number of contiguous Holding Registers to write *)

object1 : INT (* 1st Holding Register value *)

...

object8 : INT (* 8th Holding Register value *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

timeout : UINT (* Timeout [ms] *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0

Broadcast Read/write

sysMbMRTU_BroadcastFC05

Broadcast Write Single Coil - Function 05 (0x05)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 4

address : UINT (* Address of the Coil to write *)

object : BOOL (* Coil value *)

turnaroundTime : UINT (* Turnaround time [ms] *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0



invenSys
Controls

sysMbMRTU_BroadcastFC06

Broadcast Write Single Register - Function 06 (0x06)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 4

address : UINT (* Address of the Holding Register to write *)

object : INT (* Holding Register value *)

turnaroundTime : UINT (* Turnaround time [ms] *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0

sysMbMRTU_BroadcastFC15

Broadcast Write Multiple Coils - Function 15 (0x0F)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 20

base : UINT (* Address of the first Coil to write *)

no : UINT (* Number of contiguous Coils to write *)

object1 : BOOL (* 1st Coil value *)

...

object16 : BOOL (* 16th Coil value *)

turnaroundTime : UINT (* Turnaround time [ms] *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0

FREE Application Notes

code 9IS24300-1 - rel. 13.03.13

sysMbMRTU_BroadcastFC16

Broadcast Write Multiple Registers - Function 16 (0x10)

Type : FUNCTION BLOCK

Code type : EMBEDDED

Input vars num: 12

base : UINT (* Address of the first Holding Register to write *)

no : UINT (* Number of contiguous Holding Registers to write *)

object1 : INT (* 1st Holding Register value *)

...

object8 : INT (* 8th Holding Register value *)

turnaroundTime : UINT (* Turnaround time [ms] *)

pollingTime : UINT (* Polling time [ms]. 0=write on variation *)

Inout vars num: 0

Output vars num: 2

error : BOOL (* TRUE=error occurred,FALSE=no error *)

diagnostics : MbMRTUDiagnostics (* Message diagnostics *)

Extern vars num: 0